

## **Angular8 - Lezione 3**

**22/01/2020**

**Paolo Cargnin**



# Struttura del corso

## 1° lezione

Introduzione  
Componenti e data binding  
base  
+ storia dei JS frameworks

## 2° lezione

Componenti e data binding  
in applicazioni avanzate

## 3° lezione

Direttive  
+ Setup progetto del corso

## 4° lezione

Services e dependency  
injection

## 5° lezione

Routing  
Observables

## 6° lezione

Forms  
Pipes

## 7° lezione

HTTP  
Autenticazione

## 8° lezione

Ottimizzazioni e ngModules  
Interface Development  
UI frameworks

# Sommario lezione



- Direttive
- Setup dell'applicativo demo (si batte sulla tastiera)

A scelta (visto che è venerdì):

- Storia dei moderni framework

# Direttive (directives)

Hanno il solo scopo di alterare il DOM.

# Direttive (directives)

## Attribute

- Sono Attributi HTML
- Cambiano solo l'elemento a cui sono assegnate
- Vengono spesso usate con []

## Structural

- Sono Attributi HTML
- Possono cambiare parti di DOM che non sono associate all'elemento Host
- Vengono usate con \*
- Se ne può usare solo uno per elemento

# Direttive (directives)

Attribute

Structural

Entrambe hanno senso di essere usate  
con il data binding

# Direttive già incontrate

## Attribute

- [ngClass]
- [ngStyle]

## Structural

- \*ngIf
- \*ngFor

# Direttive già incontrate

Attribute

Structural

Possiamo crearne di nostre  
(custom)



# Components e data binding

- Far comunicare i componenti tra di loro:  
@Input e @Output
- Leggere i valori degli elementi HTML  
quando necessario:  
Local references e @ViewChild
- Usare ngContent

# Ciclo di vita e hook di un componente

`ngOnChanges`

Chiamato quando una input cambia

`ngOnInit`

Chiamato quando il componente è inizializzato

`ngDoCheck`

Chiamato ogni volta che un evento viene eseguito ((input),(click),(change) ecc ecc)

`ngAfterContentInit`

Chiamato quando il contenuto (ng-content) è stato parsato e renderizzato nella view

`ngAfterContentChecked`

Chiamato ogni volta che il contenuto è stato controllato, anche se non cambia

`ngAfterViewInit`

Chiamato dopo che il contenuto è stato parsato e renderizzato

`ngAfterViewChecked`

Chiamato ogni volta che il contenuto è stato controllato, dopo che è stato renderizzato, anche se non cambia

`ngDestory`

Chiamato quando il componente è stato rimosso

**Tempo di provare un po' tutto!**

# Creiamo un'applicazione con queste caratteristiche:

1. La app avrà una lista di ricette
2. L'utente potrà selezionare la ricetta da fare scegliendone una dalla lista
3. Una volta letta la ricetta, l'utente potrà scegliere di mettere nella sua lista della spesa gli ingredienti per quella ricetta

# Procediamo come segue

1. Creiamo il mockup logico dei componenti
2. Capiamo cosa possiamo fare in base alle nostre attuali conoscenze e decidiamo come strutturare il codice
3. Scriviamo!

# Storia dei moderni framework

1. JQuery – Semplificare JavaScript
2. Framework che seguono un PATTERN – AngularJs
3. Virtual DOM e componenti
4. Webpack a chiudere il tutto

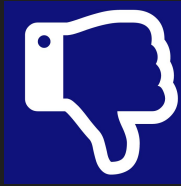
# JQuery - Semplificare JavaScript



- Selezione
- Manipolazione del DOM
- Gestione degli Eventi

```
//close modal & deatroy carousel
$('.close-gallery').click(function() {
  $('.modal-gallery').addClass('modal-gallery-close');
  setTimeout(function(){
    $('.recive-gallery').slick('destroy');
  }, 500);
});
```

# JQuery – Semplificare JavaScript



- Fetch di chiamate AJAX
- Gestione degli INPUT (validazione, getter e setter)
- Gestione di eventi poco performante (scroll, resize, view update)
- Gestione delle variabili Globali



# JQuery - Semplificare JavaScript



```
$('.nl-form').each(function(){
    var privacyCheck = $(this).find('#privacy_check');
    var txtNormal = privacyCheck.parent().find('.text-white.small').html();
    var messagePrivacy = $('html').attr('lang') === 'it-IT' ? 'Checkbox obbligatoria' : 'Mandatory checkbox';
    var cityreuire = $(this).find('input#city').length > 0;
    var professionreuire = $(this).find('input#profession').length > 0;
    $(this).validate({
        rules: {
        },
        messages: {
        },
        // grecaptcha.ready(function() {
        submitHandler: function(form,e ) {
            e.preventDefault();
            if (! privacyCheck.prop('checked') == true){
                privacyCheck.parent().find('.text-white.small').html(`${txtNormal} - ${messagePrivacy}`);
                return false;
            }else{
                privacyCheck.parent().find('.text-white.small').html(txtNormal);
            }
            $(form).parent().addClass('loading');
        },
    });
});
```

# Framework che seguono un PATTERN - AngularJs



- Two way data binding tra CONTROLLER e DOM
- Gestione di variabili globali e locali
- Template Engine integrato
- Gestione degli url (Routing)

```
<!doctype html>
<html ng-app>
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.8/angular.min.js"></script>
  </head>
  <body>
    <div ng-controller="myApp">
      <label>Name:</label>
      <input type="text" ng-model="yourName" placeholder="Enter a name here">
      <hr>
      <h1>Hello {{yourName}}!</h1>
    </div>
  </body>
</html>
```

```
angular.module('todoApp', [])
  .controller('myApp', function() {
    this.yourName = 'Paolo Cargnin'
  });
```

# Framework che seguono un PATTERN - AngularJs



- Gestione di eventi poco performante (scroll, resize, view update)
- Manipolazione del DOM
- Applicativo globale, da assegnare a tutta la soluzione

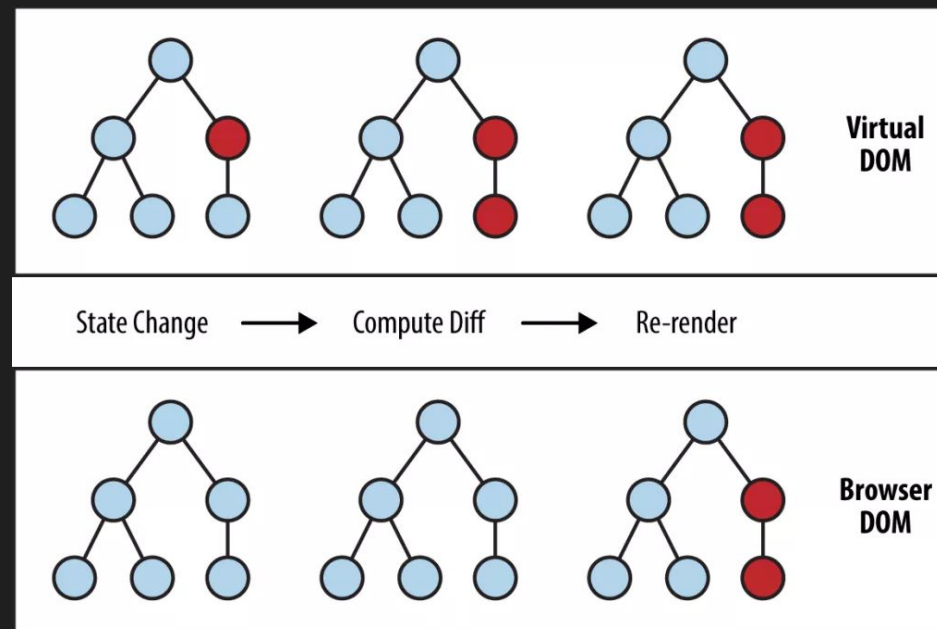
Inoltre: Sarebbe impossibile pensare insegnare angularJs nelle scuole.

```
angular.module('todoApp', [])  
  .controller('myApp', function() {  
    this.yourName = 'Paolo Cargnin'  
    $('input[ng-model="yourName"]').val('Giamba Gennari'  
  
    console.log(this.yourName) // -> Paolo Cargnin  
  
    $timeout(function () {  
      console.log(this.yourName) // -> Giamba Gennari  
    });  
  });
```

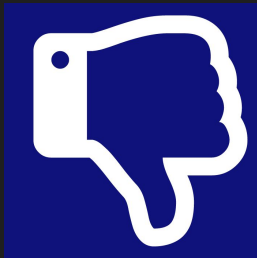
# Virtual DOM e componenti - REACT



- ONE way data binding
- Manipolazione del DOM efficace
- Tutti i vantaggi di un framework per applicativi complessi
- Può essere usato anche in porzioni della pagina web



# Virtual DOM e componenti - REACT



- Curva d'apprendimento
- Una dipendenza specifica per ogni compito (axios, react-router, Immutable ecc ecc)

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.scss';
import App from './components/App/';

import registerServiceWorker from './registerServiceWorker';

import { createStore, applyMiddleware } from 'redux';
import thunk from 'redux-thunk';
import { loadTranslations, setLocale, syncTranslationWithStore } from 'react-redux-i18n';
import reducers from './reducers';

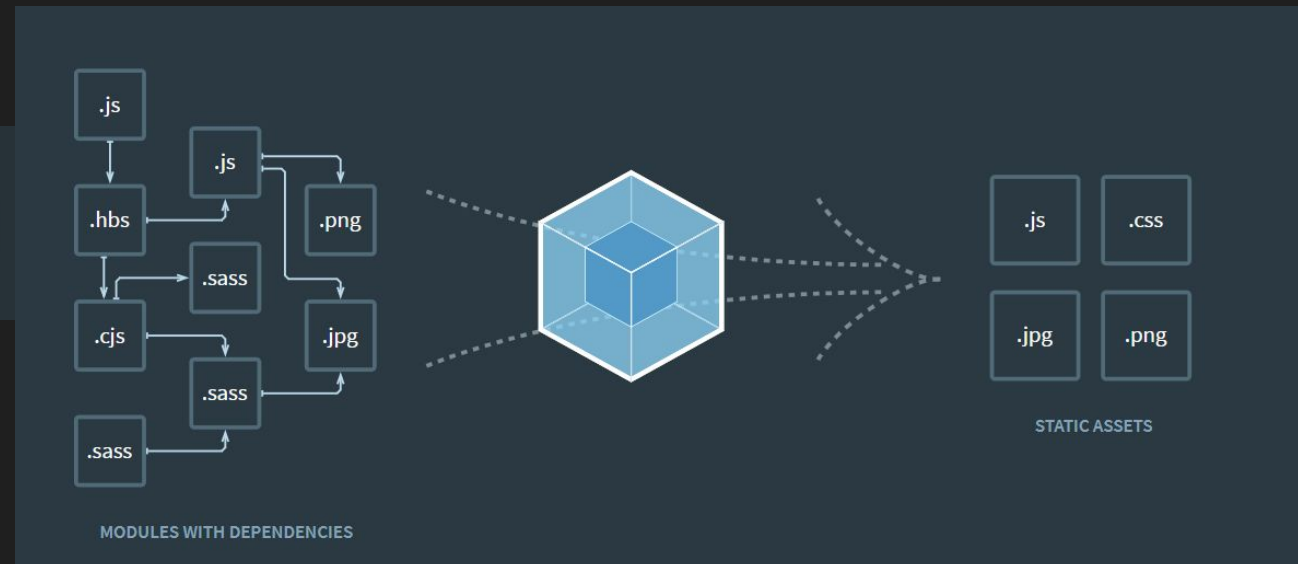
import translationsObject from './languages';
import { Provider } from 'react-redux';
```

# Webpack, export (default e named) e import

```
import SlideViewer from './SlideViewer.vue'

import '@assets/presentations/js/plyr.min.js'
import '@assets/presentations/scss/plyr/plyr.scss'
import '@assets/presentations/scss/main.scss'

export default {
  export const postHeaderConfig = {
    headers: {'Content-Type': 'text/xml'}
  };
}
```



# Rimangono ancora un po' di cose migliorabili

- La gestione delle dipendenze è un po' confusionale
  - Inizializzare un progetto non è esattamente semplice
- Validare e conoscere tutti i framework e relativo utilizzo è impossibile

# Diversi tipi di soluzioni con diversi risvolti

- TypeScript
- Angular Cli, Vue Cli ecc
  - Babel
  - JSX
  - ecc ecc